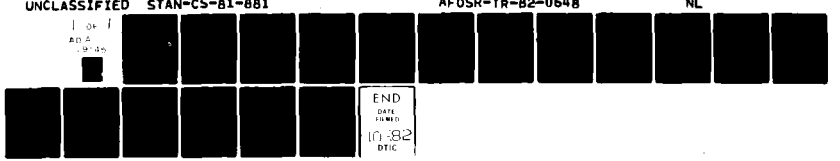
AD-A119 145 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 5/2
THE U.R. STRIKES BACK.(U)
JUL 82 J D ULLMAN.
UNCLASSIFIED STAN-CS-81-881 AFOSR-80-0212 AFOSR-TR-82-0648 NL



END
DATE FILMED
10-82
DTIC

October 1981

Report. No. STAN-CS-81-881

AFOSR-TR-82-0648

The U. R. Strikes Back

by

Jeffrey D. Ullman

AFOSR-80-0212

Department of Computer Science

Stanford University
Stanford, CA 94305

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0648	2. GOVT ACCESSION NO. AD-A119145	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE U.R. STRIKES BACK		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jeffrey D. Ullman		8. CONTRACT OR GRANT NUMBER(s) AFOSR-80-0212
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science Stanford University Stanford CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE Jul 1982
		13. NUMBER OF PAGES 13
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper, the authors try to put to rest many of the objections to the universal relation concept that have appeared in the literature. First, they shall taxonomize the varieties of ideas that are sometimes called the 'universal relation assumption'. Then, they shall consider some of the arguments pro and con. In some cases, the arguments against were expressed prematurely, and solutions to the problems they expose have since been found. In other cases, the arguments against are simply fallacious. In still other (CONTINUED)

ITEM #20, CONTINUED: cases, the problems pointed out are real, but simply serve to point out that the advantages of the universal relation are not gotten for free. The authors shall conclude the paper with a description of the algorithm used to interpret queries in System/U, and the reasoning behind it.

THE U. R. STRIKES BACK

Jeffrey D. Ullman†
Stanford University

ABSTRACT

In this paper, we try to put to rest many of the objections to the universal relation concept that have appeared in the literature. First, we shall taxonomize the varieties of ideas that are sometimes called the "universal relation assumption." Then, we consider some of the arguments pro and con. In some cases, the arguments against were expressed prematurely, and solutions to the problems they expose have since been found. In other cases, the arguments against are simply fallacious. In still other cases, the problems pointed out are real, but simply serve to point out that the advantages of the universal relation are not gotten for free. We shall conclude the paper with a description of the algorithm used to interpret queries in System/U, and the reasoning behind it.

I. The Universal Relation Assumptions

We can identify five different ideas that have at various times and places gone under the name "universal relation assumption."

1. The idea that all the attributes are initially available for the purpose of arbitrary combination into relation schemes as we do a database scheme design has been used, for example, in [B]. The underlying assumption is that we have done sufficient renaming of attributes that a unique relationship exists among any set of attributes, and the assumption is occasionally criticised for this reason. Let us call this version the *universal relation scheme* (UR Scheme) assumption.
2. There is the notion, which appears to be tacitly present in [C], but which is surely found in [ABU], that an important criterion to be satisfied by a database scheme is that it possess a *lossless join*, i.e., when a hypothetical universal relation is projected onto the relation schemes, and the projections are joined, the original universal relation results. Let us call this assumption the *universal relation/lossless join* (UR/LJ) assumption. Notice the emphasis on the word "hypothetical" describing the universal relation connected with the UR/LJ assumption. The UR/LJ assumption does not require that the universal relation actually exist, or even that it be meaningful. We merely assert that by thinking about this hypothetical universal relation, one can get some clues as to how the database scheme should be designed.
3. There is an assumption, in [HLY], e.g., that the database system should strive to maintain a collection of relations that are the projections of some one universal relation. This assumption, called the *Pure UR* assumption, is one that I shall not defend, although the questions it raises are interesting.
4. There is the assumption from [FMU] that the universal relation may be assumed to satisfy a single join dependency and a collection of functional dependencies. In particular, any multivalued dependencies that hold will follow logically from the join dependency. There is a lot of power that stems from this assumption, as we shall see when we get to the question of how System/U interprets queries. In particular, it provides us with the mechanics to interpret arbitrary queries about the universal relation as if that relation actually existed. However, I am not willing to defend it categorically. As far as I would go is to say that with the addition of user-defined maximal objects [MU1] to simulate embedded multivalued dependencies that do not follow from the join dependency, there is a good chance that whatever semantics for the UR the user wishes will be definable in these terms. Let us agree to call this the UR/JD assumption.
5. A strengthening of the UR/JD assumption is that the join dependency that the UR satisfies must be acyclic in the sense of [FMU]. Call this the *Acyclic JD* assumption. Among other benefits, there is a strong sense, described in [MU2], in which an acyclic join dependency implies a unique interpretation of queries over the universal relation. It appears that this assumption is too strong to hold in general, although by renaming attributes that are used in two different senses, it becomes possible to make the structure acyclic in more instances than detractors might imply. Further, when the UR structure is

† Work partially supported by AFOSR grant 80-0212 in accordance with NSF agreement IST-80-21358 and by NSF grant MCS-80-12907.

not acyclic, it is possible to construct a collection of acyclic maximal objects that provide most of the benefit of the Acyclic JD assumption at a penalty of making it more difficult to express certain queries than it would be under the UR/JD assumption.

II. The Case for the Universal Relation View

Before considering in detail the criticisms of the various universal relation assumptions, let us review why the author believes the concept worth considering in the first place. The merit of the concept is that it allows the user to query a database as if there were a single relation. This relation may have nulls in certain components of certain tuples, and these nulls should be marked, that is, all nulls are different, unless equality follows from a given functional dependency. For example, if we don't know Jones' address then there is a symbol that stands for "the address of Jones" in every tuple of the universal relation in which that address should logically appear, and in no others. Remember that this universal relation doesn't actually exist, except in the user's mind, so the nulls may not appear in the actual database.

The purpose of allowing the user to see the database as a single relation is that he is thus relieved of the need to learn about many details of the database structure.

Example 1: Suppose we have attributes E (employee), M (manager), and D (department). The user should be able to say something like

retrieve(D)
where E ='Jones'

without concern for whether there is a single relation with scheme EDM, or two relations ED and DM, or even EM and DM. []

Let us not infer from Example 1 that no knowledge of the database's semantics is required of the user. It is a matter of taste whether understanding concepts like "employee" and "department" and their expected relationship is much easier on the user than understanding what an ED relation means. I think there is some gain in intuition to be had by the universal relation viewpoint, and surely, many queries are easier to express in this way than in a query language that deals with individual relations.

There is more empirical evidence that the universal relation user view justifies the attention given to the concept.

1. There has been a history of success in a variety of database applications using Brian Kernighan's system/q at Bell Laboratories [A]. This system supports a universal relation by means of a *rel file*, which is a list of joins that could be taken if the query requires it; the first join on the list that covers all the needed attributes is taken. If there is no such join on the list, the join of all the relations is taken.
2. A variety of natural language systems, such as [C*], [DSK], and [Mo], tacitly use the universal relation as a user view. Indeed it is hard to see how a natural language system could reliably use anything else, although certain words could, and are, used to infer that certain relations are being talked about.

Let us therefore take as a point of departure that there is some utility in considering a system that supports the universal relation as a user view. We are not saying that every database should be implemented in this way, or even that it could; we just contend that there is enough of a chance that the concept will be *useful in a given context, and enough* benefit to the user if it is, that the approach should be considered.

If we assume the universal relation as a user view, then we must make the UR/LJ assumption when we design the actual database. The reason is that if we do not have a lossless join for our database scheme, then the database will not represent a unique universal relation [ABU], which is strong evidence that the database is not adequate for the task we have assigned it. We also must make the UR/JD assumption; the justification is given in [FMU]. We cannot assume that the join dependency we get will be "interesting" though. The assumption that it usually has a great deal of structure, that is, many terms of a few attributes each, is a matter of belief. Obviously, we have also made the UR Scheme assumption, but the other two assumptions, the Pure UR and Acyclic JD assumptions, have not been made.

III. Attacks on the Universal Relation Assumptions

Let us now consider a nonexhaustive list of the arguments that have been used against one or more of these assumptions. We maintain that, in a sense, neither the arguments nor their refutations are important. The fact that systems using the concept exist and are successful, even in a limited context, is a more powerful

refutation of the arguments against the universal relation idea than are any convolutions of logic that the author can provide.

The Pure UR assumption is responsible for certain inadequacies of Boyce-Codd normal form.

This viewpoint, from [BG], is an example of a premature attack on the UR concept. The problems mentioned by [BG] center around the inability of the authors to think of a way to do updates to universal relations in the presence of null values. This issue is certainly important if we are to support the universal relation as a user view, and [BG] is quite correct in pointing out that a problem, not completely resolved today, exists.

Unfortunately, in analyzing the problem, the authors use only a single null value, and make an unfounded assumption that when inserting, a tuple that is more defined than another causes deletion of the less defined tuple. In contrast, at the same time [BG] was being written, works such as [KU] and [Ma] were developing a semantics of nulls that assumed all nulls were different and could be made equal only if it followed from given dependencies. One error [BG, p. 253] makes is assuming that

> "The correct action apparently is to replace <null, null, g> by <v, 14, g> [which is another tuple in the relation being talked about]."

in a situation where the third component does not functionally determine either of the other components. Their "correct action" is incorrect because there is no logical justification for why the first null equals v or the second equals 14.

Another problem of [BG] concerned deletions. The authors did not have available to them the deletion strategy of [Sc], which replaces a deleted tuple t by all tuples that have the components of t in proper subsets of the nonnull components of t, and nulls elsewhere (there is also the constraint that the nonnull components must be an "object" in the sense of [Sc], i.e., have meaning as a unit). Indeed, not all deletions are permitted by [Sc], on the grounds that certain ones do not make sense, but his theory is consistent, and it does explain away many of the objections of [BG].

Incidentally, I believe that the problems with BCNF are not caused by the universal relation assumption in any form. Rather the problem is that the violating dependencies are observations that follow from the "physics" of the situation, but contribute nothing to the database structure. They should simply be ignored.

The work of [BG] does point out the need for a consistent theory of updates for a universal relation system. It is probably not completely satisfactory to do, as system/q does, all updates as processes on files separate from the query system itself. However, the works cited on semantics of nulls do provide a reasonable theory of updates for universal relations. Work on the issues remains to be done, and it is an important open question to find the limits on our ability to update universal relations in a meaningful fashion.

Because attribute splitting is necessary to have the UR Scheme assumption, attribute names will in effect have their relation attached to them anyway.

This view is expressed by [Ke], for example. It is true that we might like to use NAME for the names of employees, customers, and suppliers, and the UR Scheme assumption forces us to use E_NAME, C_NAME, and S_NAME, for example, and it is also true that this usage is not an improvement over something like EMPS.NAME, CUST.NAME and SUPS.NAME.

However, the assumption that every attribute, or even most attributes, must be modified this way does not seem to be borne out in practice. The criticism seems to make the underlying assumption that unless a new methodology is always better than the old, it is of no use at all. Furthermore, the criticism misses the point in assuming that the only thing the universal relation idea is after is a short way of referring to attributes. Far more important than this is the transparency of connections in the database. I don't mind admitting that the name I am talking about is a customer's name when I write

```
retrieve(BILLS)
where C_NAME='Jones'
```

if I can be spared the burden of explaining to the system that customers are connected to bills by looking up the customer's credit card number in one relation, finding the sales slips with that number on them in another relation, and then finding the invoices referring to each of those sales in a third relation.

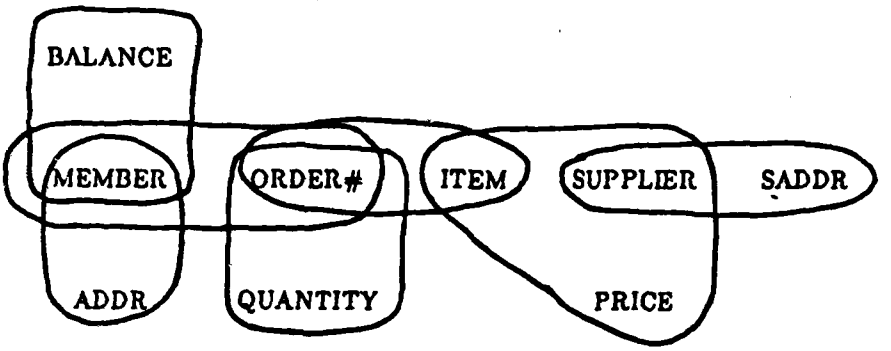


Fig. 1. The HVFC Database.

The UR/LJ assumption is nothing more than defining a view—one that is the natural join of all the relations.

There is an important difference concerning the way queries are interpreted. We shall say more about query interpretation in Section V, but for the moment, let us suggest what the difference is with a simple example.

Example 2: In Fig. 1 we see the hypergraph representation of the objects (minimal, logically connected sets of attributes) in the Happy Valley Food Coop example from [U]. The relations of the database would probably be supersets of some of these objects. For example, MEMBER, ADDR, and BALANCE would probably be grouped in one relation, ORDER#, QUANTITY, ITEM, and MEMBER in another, SUPPLIER and SADDR in one, and SUPPLIER, ITEM, and PRICE in a fourth.

We could define a view that was the natural join of these four relations and make queries about this view. However, consider a query like

```
retrieve(ADDR)
where MEMBER='Robin'
```

If, say, Robin had placed no orders, or he had placed orders, but not for items that had suppliers at the moment, the natural join view would have no tuples with MEMBER='Robin', and we would get no address in response.

However, if we use the System/U interpretation of queries, from Section V, we discover that all but the MEMBER-ADDR object is superfluous, and we interpret the query as the obvious one on the MEMBER-ADDR-BALANCE relation. The difference is that if we define the natural join view, a standard system is required to use *strong equivalence* in simplifying the query, meaning two expressions are considered equivalent if and only if they produce the same answer for arbitrary relations. Since missing tuples, such as no orders for Robin, make the selection and projection on the view and on the single relation different, a standard system cannot optimize this query. On the other hand, System/U makes the Pure UR assumption when optimizing queries, that is, it uses the *weak equivalence* criterion of [ASU1]. The two formulations of the query (one on the natural join, the other on the MEMBER-ADDR-BALANCE relation only) are weakly equivalent, so System/U produces the intuitively correct answer. □

We claim the System/U answer is more likely to be correct than the answer that would be obtained from the view, because if we ask only about Robin's address we probably don't care about any orders he placed.† We should emphasize that the use by System/U of the Pure UR assumption for query optimization is in a sense a "kludge." The reader must judge if the results, especially the use of weak equivalence to optimize queries, justify the assumption. The reason we believe it justifiable is that dangling tuples, even though we admit they exist in typical physical relations, should have no part in the answer if the query doesn't really need their relation. Thus, the descrepancies between what would happen if the relations were the projection of a universal relation and what happens in fact, makes no difference in the intuitively correct answer.

The Acyclic JD assumption fails to meet the intuitive criteria for absence of cycles.

This claim, expressed in [AP], appears to depend on identifying two hypergraphs that we do not consider interchangeable. In Fig. 2 we see the banking example from [FMU], and in Fig. 3, we see what happens when [AP] redefines the objects, replacing the BANK-ACCT and ACCT-CUST objects by their union and

† If we do care, we can force the order number to be considered by adding a term like ORDER#=ORDER# to the where-clause.

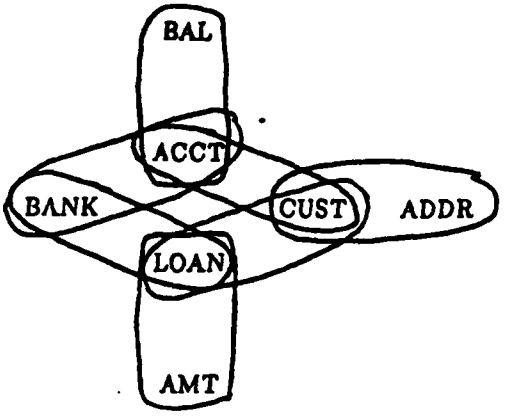


Fig. 2. The banking example.

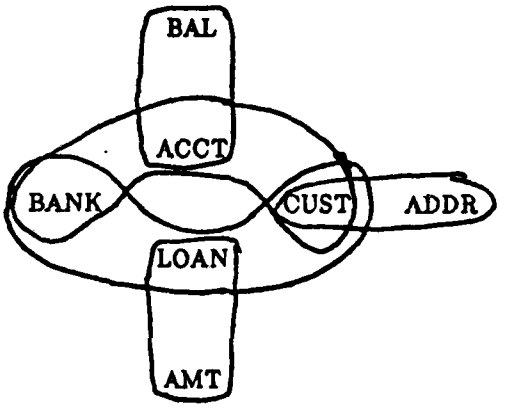


Fig. 3. The banking example with different objects.

doing the same thing with LOAN. Figure 3 is acyclic in the sense of [FMU], as it should be, because if the hypergraph were drawn differently, as in Fig. 4, the "hole" disappears.

However, even if that were not the case, [AP] is wrong in assuming that the hypergraphs of Figs. 2 and 3 are related. If one consults [FMU] for the way hypergraphs are related to our assumptions about the real world, we see the difference between the two figures. In Fig. 2, customers are related to banks through accounts and loans. If two customers share an account at Bank of America, then both are related to Bank of America. However, Fig. 3 presents a different view of the real world. It says that BANK-ACCT-CUST is a fundamental relationship, so two customers can share an account at two different banks, and each will be related to only one of the banks. Now you can guess which the author thinks is the real "real world," but which is correct is irrelevant; what is important is that they are different.

There is another issue raised by [AP] when they justify their claim that Fig. 3 is "cyclic" by pointing to the definition of an acyclic Bachmann diagram in [L]. It is well known [FMU] that the two notions of acyclicity are different. In addition to the remarkable properties known for acyclicity in the [FMU] sense, as enumerated in [B*], there is a very practical justification for using the [FMU] definition in the Acyclic JD assumption. If we minimize a query about an acyclic universal relation scheme, the resulting set of objects to be joined should in some sense lie between the attributes mentioned by the query. The objects in the join should also include all those that lie on the minimal paths connecting the attributes of the query. In [MU2] a result of this nature is shown to apply to exactly the acyclic hypergraphs in the [FMU] sense.

Interestingly, [Y] shows a stronger result about the connections in those hypergraphs that are acyclic Bachmann diagrams, but it is questionable whether we need the extra strength; that is, the connections obtained in [MU2] seem adequately strong and unique. Again, the matter of which definition of "acyclic" you prefer is one of taste, but one should not confuse the two notions. In fact, [F] discusses three distinct

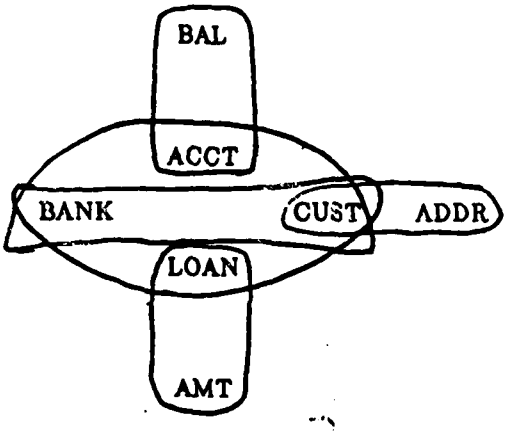


Fig. 4. View of Fig. 3 showing acyclicity.

notions of acyclicity, including the two mentioned here, and compares their vitues.

The acyclic JD assumption is unreasonable because most real world databases have an inherently cyclic structure.

This viewpoint is also from [AP]. I'm not sure whether I believe that or not. Surely the ability to split attributes helps us turn cyclic structures into acyclic ones, but the cost is often high—simple queries must be circumlocuted, by using an equijoin to connect the two formerly identical attributes. However, splitting is only one option. The maximal objects concept from [MU1] can be used to identify the acyclic substructures of a cyclic one and to allow navigation through the acyclic structures automatically.† The only time one is forced to use equijoins to connect seemingly identical items is when one's query jumps among acyclic structures. But it is exactly at these times that the paths connecting attributes become nonunique, and the extra specification of path is essential.

Example 3: In Fig. 5 we see the example from [AP], which they attribute to [Mc], of a "real world." That entity-relationship diagram is translated into objects in Fig. 6. The objects have been numbered for reference, and the functional dependencies implied by the many-one relationships of Fig. 5 have been shown by arrows. Since all objects are of size two, the ones that are not functional dependencies are indicated by lines rather than ovals as in the usual hypergraph representation. Certain of the one-one relationships that appear to be "isa," or subset relationships, have been shown as functional dependencies from the subset to the superset only. This treatment of "isa" simplifies the construction of maximal objects and adapts a suggestion of C. Beeri that "isa" be followed only from subset to superset when constructing maximal objects. It is not, however, essential for the point to be made. We have also assumed that the relationship between sales and customers is really the composition of the relationships from sales to orders to customers. We have thus eliminated one of the cycles present in Fig. 5, but this elimination is not essential in what follows.

If we build maximal objects as suggested in [MU1], by starting with single objects and adjoining additional objects if the lossless join of that object with what is already included follows from the functional dependencies given or from those multivalued dependencies that follow from the given join dependency (there are no useful dependencies in this category for this example), then we get the following five maximal objects.

$$M_1 = \{1, 2, 3, 4, 6, 7, 8\}$$
$$M_2 = \{5, 8, 9, 10, 11, 12\}$$
$$M_3 = \{8, 9, 10, 13, 15, 18\}$$
$$M_4 = \{8, 9, 10, 14, 16, 17\}$$
$$M_5 = \{8, 9, 10, 19, 20\}$$

These can be constructed starting with objects 4, 5, 18, 16, and 19, respectively.

As an example of the power inherent in this view of the database, we could answer a request from a customer to verify the deposit of his check by a query like

† Depending on which of several rules are used to construct maximal objects, they may or may not be guaranteed to be acyclic.

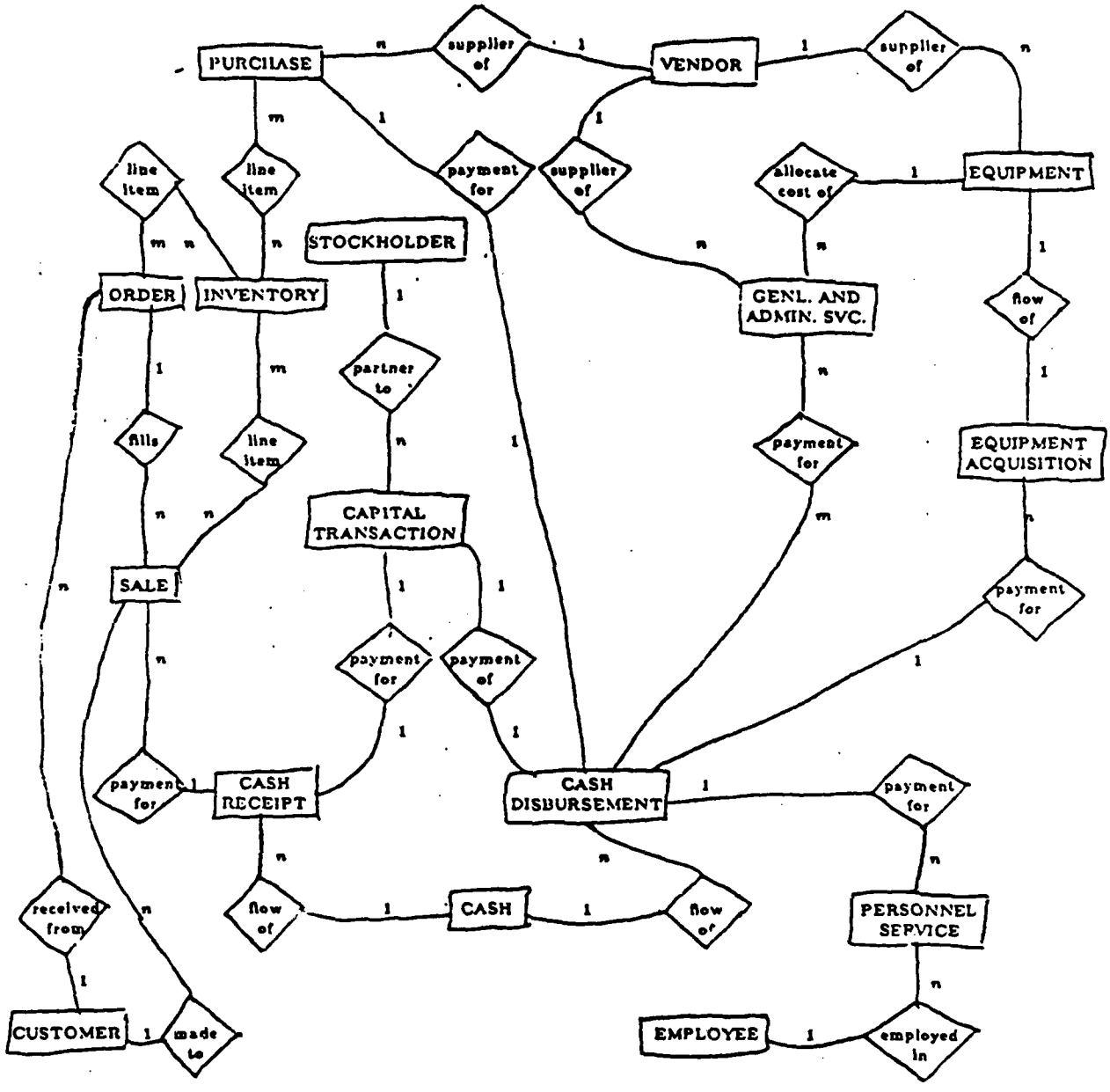


Fig. 5. Retail enterprise "real world."

```
retrieve(CASH)
where CUSTOMER='Jones'
```

This query is perfectly natural, and causes the system to navigate through several objects, and probably through several relations, in the M_1 maximal object.†

As another example, the query

```
retrieve(VENDOR)
where EQUIPMENT='air conditioner'
```

is answered by giving the union of the vendors connected to the air conditioner either through "general and administrative service" in the maximal object M_3 or through equipment acquisition in M_4. That is a reasonable response to this ambiguous query, and follows the strategy suggested by [Cha, O, Sa1, Sa2], for

† Section V. discusses the System/U treatment of queries; the reader can take the description given here for granted.

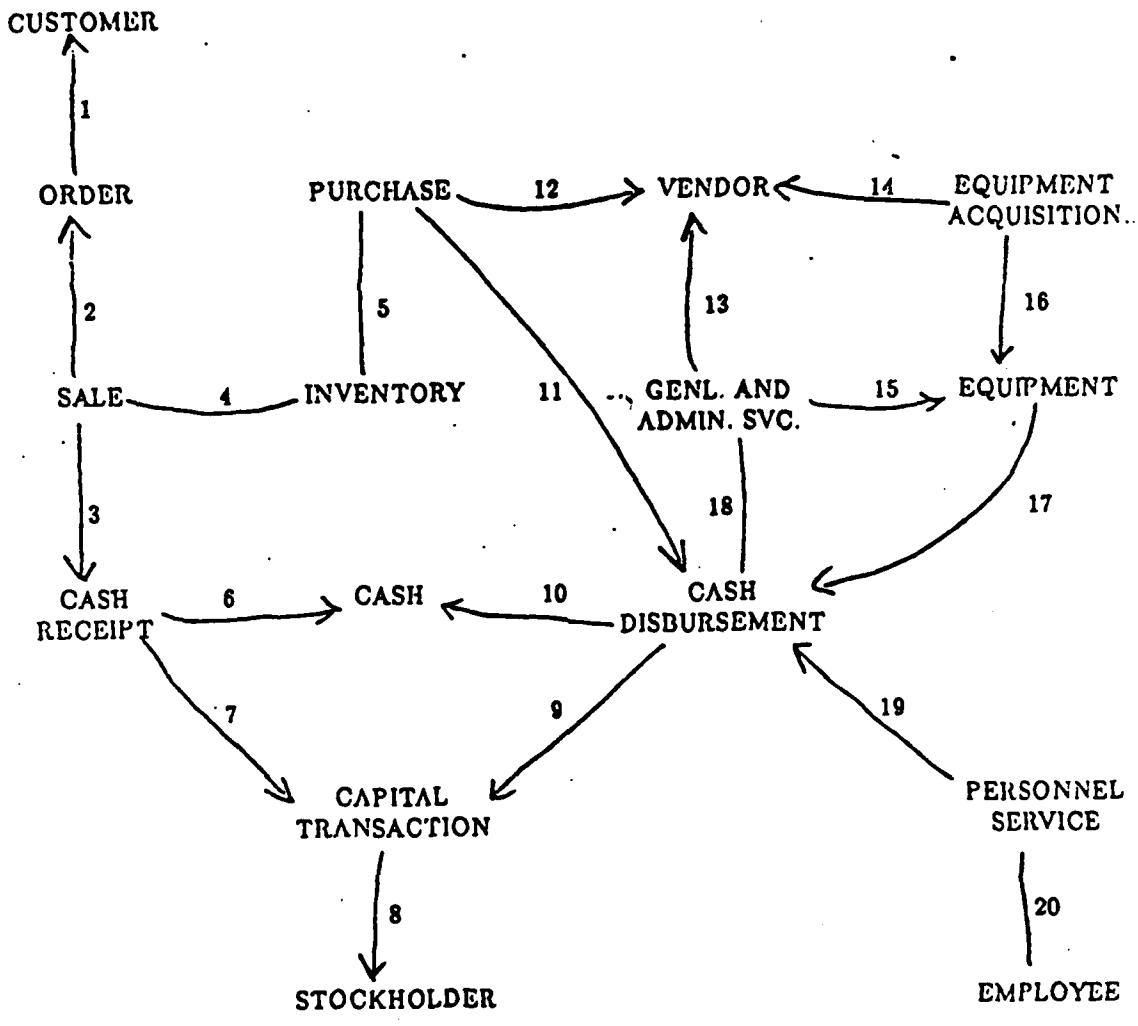


Fig. 6. Hypergraph for retail enterprise.

example. []

There is no way for the user of a universal relation system to know whether the answer to his query is the one he expected.

The technique of having the system paraphrase the query, the way many natural language systems do, would probably be of some help here. However, the fallacy in leveling this charge at the universal relation concept is not that no problem exists; rather, the problem is not restricted to universal relation systems. For example, [GW] reports an experiment with teaching college students to use relational query languages, in which after training, they still had error rates like one out of three in expressing queries that involved joins. If you believe that is typical of naive, but presumably intelligent users, then it is hard to assert that ordinary database systems always give the user what he expected. In fact, since the study of [GW] implies that queries needing joins were considerably harder for students to get right than were queries involving only one relation, there is hope that a universal relation system would give them much lower error rates than ordinary systems.

Universal relation systems fail to take into account all possible connections among the attributes involved in a query.

This point has figured into the thinking of [AP], where they discuss the "relationship uniqueness" problem,

and is implicit in the approach of [Sa1, Sa2] in dealing with universal relations. While there are arguments on both sides, my own point of view is that all relationships are not equally plausible as query interpretations. I conjecture that a carefully designed system can deduce the user's intent, from among possible connections, with high enough accuracy to make the system workable. As we remarked in the subsection above, ordinary systems are not guaranteed to "interpret" queries correctly for the simple reason that the complexities of ordinary query languages make user errors frequent.

The following is an example of a situation where making distinctions between connections seems quite reasonable. In Fig. 2, a query like

```
retrieve(LOAN)
where CUST='Jones'
```

is most likely to be asking for the information in the CUST-LOAN object, not for the connection between CUST and LOAN that goes through ACCT and BANK, to give those loans made by a bank at which Jones has an account. The query language should allow the latter connection to be specified if the user desires it, but it appears to the author quite reasonable to take the simpler connection as a default in the absence of a specification to the contrary.

The process of selecting interpretations for queries over a universal relation is still some art and some science. We feel, however, that progress has been made, and there is hope for more progress in the future.

IV. The System/U Data Definition Language

Now let us sketch the design of System/U and see how the ideas discussed above can be made to work in practice. The design of the system uses ideas contributed by H. Korth and G. Kuper, as well as those of the author. Earlier descriptions appear in [KU, Ko]. The data definition language includes the following kinds of declarations.

1. Attributes and their data types.
2. Relation names and their schemes (sets of attributes).
3. *Functional dependencies.*
4. Objects, which are sets of attributes, and the relation from which each object is taken, with possible attribute renaming allowed.
5. Maximal objects, which are sets of objects.

Points (4) and (5) require some elaboration. Objects are the edges of the hypergraph that defines the join dependency assumed to hold in the universal relation. They are, intuitively, the minimal sets of attributes that have collective meaning, and the term is taken from [Sc], where the concept was first developed. For example, in terms of the entity-relationship model [Che], an attribute or attributes that form a key for an entity set will be found in one object for each of the properties of that entity set; the object includes only the key and the one property. Relationships are represented by objects consisting of the keys for the related entity sets.

Each object is assumed to be contained in one relation, perhaps properly contained if the relation is unnormalized, or if the relation consists of a key and many properties of that key. We allow renaming of attributes so that the same relation can be used for many objects that are effectively identical.

Example 4: A genealogy can be based on a single relation CP, the child-parent relationship. We might declare attributes PERSON, PARENT, GRANDPARENT, AND GGPARENT, with objects PERSON-PARENT, PARENT-GRANDPARENT, AND GRANDPARENT-GGPARENT, each defined to be the CP relation with the obvious correspondence of attributes. We could then ask

```
retrieve(GGPARENT)
where PERSON='Jones'
```

and find the great grandparents of Jones in the obvious way, taking what the system thinks are natural joins, but are really equijoins on the CP relation.

As another example, we could make the banking database of Fig. 2 acyclic (a step we do not recommend) by splitting CUST into DEPOSITOR and BORROWER and splitting ADDR into CADDR and BADDR. We would then use objects ACCT-DEPOSITOR, LOAN-BORROWER, DEPOSITOR-DADDR, and BORROWER-BADDR. One problem with this approach (in addition to forcing the user to remember the unimportant difference between BADDR and DADDR, which is why we do not reccommend the split) is that

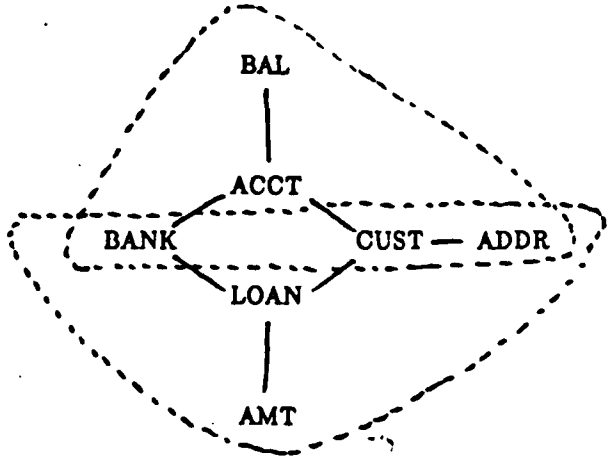


Fig. 7. Maximal objects in the banking example.

we appear to need two relations recording names and addresses, and many of these will be duplicates, when a customer is both a borrower and depositor. However, we can create one relation of names and addresses and declare both objects DEPOSITOR-DADDR and BORROWER-BADDR to be this relation, which alleviates at least one problem. []

The fifth item above, declaration of maximal objects, also requires some explanation. The system computes maximal objects itself, using the functional dependencies and multivalued dependencies implied by the join dependency on the objects.† However, the user can override the automatic computation by declaring additional maximal objects. The system then throws away those of the maximal objects it computes that are subsets or supersets of the declared objects. One important use of this feature is in simulating embedded multivalued dependencies, which have no place in System/U otherwise.

Example 5: Suppose in the banking example of Fig. 2 we decide that the functional dependencies

$$\text{ACCT} \rightarrow \text{BANK},\ \text{ACCT} \rightarrow \text{BAL},\ \text{LOAN} \rightarrow \text{BANK},\ \text{LOAN} \rightarrow \text{AMT},\ \text{and}\ \text{CUST} \rightarrow \text{ADDR}$$

hold. Then the two maximal objects shown in Fig. 7 would be constructed.

A query like

```
retrieve(BANK)
where CUST='Jones'
```

would give the banks at which Jones has either a loan or account, since CUST is connected to BANK in two different ways through the two maximal objects.

However, suppose we denied the functional dependency LOAN→BANK. That means, intuitively, that loans can be made by consortiums of banks. The lower maximal object in Fig. 7 is now replaced by two, BANK-LOAN-AMT, and CUST-ADDR-LOAN-AMT. If we ask the query above, we get only the banks at which Jones has accounts, because only the top maximal object connects CUST to BANK now.

Perhaps the reader feels that this answer is satisfactory, because the connection between BANK and CUST through LOAN is now "weaker" than through ACCT. More likely, the reader feels that the connection through LOAN is still just as valid as that through ACCT. Intuitively, the reader probably has in his mind the model in which each bank in a consortium has made the loan to each borrower of that loan. If that is the case, then there is an embedded multivalued dependency LOAN→→BANK | CUST. It turns out that the practical effect of this multivalued dependency can be achieved by declaring the lower maximal object of Fig. 7 to hold, even though it won't follow from the given functional dependencies or from the join dependency on the objects. []

V. The System/U Query Language

We shall now sketch the ideas behind the query language and its implementation. The language itself is essentially QUEL [S*], with the following important difference. Since all tuple variables range over the universal relation, there is no need for a range statement or declaration of tuple variables. Furthermore, an

† As a result, maximal objects may not be acyclic. They will always have a lossless join, however.

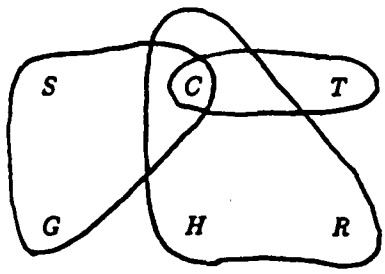


Fig. 8. Courses example.

attribute A by itself is deemed to stand for $b.A$, where b is the *blank tuple variable*, a tuple variable that never appears but is inserted when we translate queries. In the vast majority of queries, all one needs is the blank tuple variable, but we provide others so you can find out about employees that make more than their managers and other important information by queries like

retrieve(EMP)
where MGR=t.EMP and SAL>t.SAL

The translation process we use for queries is

1. For each tuple variable, including the "blank" tuple variable that we associate with attributes standing alone, assign a copy of the universal relation. Begin by taking the Cartesian product of all these copies of the universal relation.
2. Apply to the Cartesian product the selections implied by the where-clause, and the projection implied by the list of attributes in the retrieve-clause.
3. Substitute for the copy of the universal relation associated with tuple variable t (which may be the blank tuple variable) the union of all those maximal objects that include all the attributes A such that $t.A$ appears in the query.
4. Substitute for each maximal object the natural join of all the objects in that maximal object.
5. Replace each object by an expression involving the actual relations in the database. Recall that each object is the projection (perhaps with renaming of attributes) of a relation in the database.
6. The resulting expression is optimized by tableau optimization techniques [ASU1, ASU2, SY]. We both minimize the number of join terms in each term of the union and minimize the number of union terms. Each of these minimizations can be done exactly, the first by [ASU1, ASU2], and the second by [SY]. As we minimize rows of a tableau, we should remember the relation from which each row comes (although we do not refrain from mapping rows to rows that come from different relations). When the minimal tableau is reached, we can use this information to reconstruct the optimized join expression. There is an important special case where, although the minimal tableau is unique, as it must be, we can obtain it by eliminating one of several rows in favor of another. In that case, we must take the union of all the join expressions that correspond to versions of the minimum tableau with rows and relations identified in any possible way.

Example 8: Consider the database of Fig. 8, where the objects are CT, CHR, and CSG, and let the relations in the actual database be $CTHR$ and CSG; note that the first of these happens not to be normalized. This database is the courses, teachers, hours, rooms, students, and grades example from [U], and the meanings of the attributes should be obvious. Let the query in question be

retrieve($t.C$)
where S='Jones' and $R = t.R$

That is, print the courses that sometimes meet in rooms in which some course taken by Jones meets.

In step (1), we begin by using $C_1T_1H_1R_1S_1G_1$ as the copy of the universal relation corresponding to the blank tuple variable, and the same set of attributes subscripted by 2 for the copy of the universal relation for the tuple variable t. For step (2), only C_2 is mentioned in the retrieve-clause, and the selection condition is that S_1='Jones, and that $R_1 = R_2$. Thus, the algebraic expression constructed at step (2) is

$$\pi_{C_2}(\sigma_{S_1='Jones' \wedge R_1=R_2}(C_1T_1H_1R_1S_1G_1 \times C_2T_2H_2R_2S_2G_2))$$

where π stands for projection and σ for selection.

The database of Fig. 8 being acyclic, the only maximal object is the entire database [MU1]. As both t

C_1	T_1	H_1	R_1	S_1	G_1	C_2	T_2	H_2	R_2	S_2	G_2
						a_1					
b_1	b_2										
b_1		b_3	b_4								
b_1				c	b_5						
						a_1	b_6				
						a_1		b_7	b_4		
						a_1				b_8	b_9

Fig. 9. Tableau for constructed expression.

and the blank tuple variable are surely associated only with attributes that are in this one maximal object, the union at step (3) is simply this one maximal object in each case. In step (4) we substitute

$$C_1T_1 \bowtie C_1H_1R_1 \bowtie C_1S_1G_1$$

for $C_1T_1H_1R_1S_1G_1$, and make a similar substitution for $C_2T_2H_2R_2S_2G_2$ in the above expression. Then, in step (5), we replace C_1T_1 by $\pi_{C_1T_1}(C_1T_1H_1R_1)$ and $C_1H_1R_1$ by $\pi_{C_1H_1R_1}(C_1T_1H_1R_1)$ and then do the same for the objects involving $C_2, \ldots, G_2$. Figure 9 shows the tableau for the resulting expression.

In Fig. 9 we have shown the distinguished symbol by a_1, the constant 'Jones' by c, and some of the nondistinguished symbols by subscripted b's. All blank positions represent nondistinguished symbols that appear nowhere else. The optimization of Fig. 9 takes place by a straightforward application of the method of [ASU1, ASU2]. However, in System/U we make several simplifications that seem not to cause optimization to be missed very frequently, and leads to considerable efficiency. First, we treat every variable that is constrained in the where-clause as if it were a constant in the sense of [ASU1, ASU2]. These symbols effectively prevent their rows from being mapped to others in a tableau reduction. In Fig. 9, we used the constant c to enforce the constraint S='Jones' and we represent the constraint $R = t.R$ by the fact that b_4 appears in two different columns; the result is that the rows containing b_4 cannot be mapped to any other row, making b_4 in effect a constant. The algorithm of [KI] to minimize tableaux in the presence of arithmetic constraints could be used to improve our potential for optimization, although it is not clear how much benefit would be obtained in practice.

A second simplification we make is to assume that the maximal objects are acyclic (although in pathological cases, functional dependencies could produce cyclic ones), and reduce the tableau by the simple process of testing whether some one row can map to another by the process of symbol renaming as in [ASU1].

For example, in Fig. 9, the first row maps to the second if we rename b_2 to the blank in the T_1 column of the second row. Similarly, rows 4 and 6 map to 5. We cannot map row 3 to row 2 because c, being a constant, cannot map to the symbol represented by the blank in the S_1 column of row 2. Also, rows 2 and 5 cannot map to any row, because b_4 would have to become two different symbols simultaneously. The optimized tableau will retain only the second, third and fifth rows of Fig. 9 (plus the summary row with a_1 alone).

The remaining rows, 2, 3, and 5, come from relations $CTHR$, CSG, and $CTHR$, respectively. If we use the optimization strategy of [WY], say, to select an order for operations, the optimized query would be answered by a sequence of steps in which

1. we select from CSG those tuples with S='Jones', and save the set of C-values from those tuples, say C,
2. then select from $CTHR$ those tuples with C-component in C and produce from them the set R of their R-values, and finally
3. select from $CTHR$ the C-components of tuples with R-components in R. []

Example 9: The rule in step (6) for handling join terms that could come from several relations is a response to the fact that the Pure UR assumption will not in general be satisfied. For example, suppose we have relations ABC, BCD, and BE, and our query asks about B and E. After optimization, we eliminate either the row for ABC or the row for BCD, but not both.

One term in the join to be taken in the optimized query is BE, since this relation's row cannot be eliminated. The other term could come from either the row for ABC or the row for BCD. We therefore

construct the expression $(ABC \cup BCD) \bowtie BE$, to which the selection and then projection onto BE is performed. Since only B and E are involved in either selection or projection, this expression can be replaced by one of the form $\pi_{BE}(\sigma((\pi_B(ABC) \cup \pi_B(BCD)) \bowtie BE))$. In effect, the set of B-values to be joined with BE is the union of what appears in the ABC and BCD relations. If we believed the Pure UR assumption, the set of B-values in the two relations would have to be the same, but we don't, and it isn't. []

Example 10: Let us consider an example where the universal relation must be composed of the union of several maximal objects because the underlying structure is cyclic. The banking example of Fig. 7 will serve nicely. Consider the query

```
retrieve(Bank)
where Cust='Jones'
```

There is only one tuple variable, the blank, so in step (1) we create a single copy of the universal relation. In step (2) we apply to this relation the operators $\pi_{\text{Bank}}\sigma_{\text{Cust='Jones'}}$. In step (3), we see that both maximal objects in Fig. 7 include the attributes Bank and Cust mentioned by the query. Thus the query becomes

$$\pi_{\text{Bank}}\sigma_{\text{Cust='Jones'}}(\text{Bank Acct Bal Cust Addr} \cup \text{Bank Loan Amt Cust Addr})$$

Each of these maximal objects is replaced by the join of their objects in step (4). On the assumption that each of the objects in Fig. 7 is also a relation, step (5) has no effect. At step (6) we break the expression into the union of two tableaux and minimize them in the obvious ways, deleting "ears" that do not serve to connect Bank with Cust. The expression becomes

$$\pi_{\text{Bank}}\sigma_{\text{Cust='Jones'}}(\text{Bank-Acct} \bowtie \text{Acct-Cust}) \cup \pi_{\text{Bank}}\sigma_{\text{Cust='Jones'}}(\text{Bank-Loan} \bowtie \text{Loan-Cust})$$

We then check whether either term of the union is a subset of the other, but that is not the case here, and the resulting expression is the one produced by System/U. []

VI. Motivation Behind the Query Interpretation Algorithm

While many other interpretations of queries are possible, we feel that there are sound arguments in favor of the six-step algorithm proposed in the previous section. The first two steps, construction of a Cartesian product of universal relations and application of selection and projection are analogous to the way QUEL handles queries [WY].

Step (3), the replacement of the universal relation by the union of the maximal objects that are relevant to the query is a matter for judgement. As mentioned previously, the idea that the union of possible connections is what we want has been expressed by several authors. For example, the extension join method for interpreting queries [Sa2] when the only dependencies are functional ones based on a key within one object (key dependencies) takes a union of connections to interpret queries.† Step (4), the replacement of the maximal object by the natural join of its member objects, follows from the fact that the construction of maximal objects we use guarantees the lossless join property for this decomposition of the maximal object [MU1].

Step (5) is mandatory, since we must ultimately talk about relations, not objects. Finally, the optimization in step (6) is guaranteed not to change the result of the query except as dangling tuples are concerned. As we argued previously, there is good reason to think that a missing tuple in a relation that isn't on any path connecting the attributes mentioned by the query should not cause tuples to be deleted from the answer.

Acknowledgement

The author appreciates the comments made by Stott Parker on an earlier draft of this manuscript.

† An important difference between the methods, in addition to the fact that extension joins ignore connections that are not based on functional dependencies, is that Sagiv computes connections dynamically, while maximal objects are computed once for all queries. That is, once an extension join reaches far enough to cover the relevant attributes, it is not constructed further, even though doing so might enable it to include another extension join. J. Gischer points out the following example. Suppose the relation schemes are AB, AC, and BCD, with functional dependencies $A \rightarrow B$, $A \rightarrow C$, and $BC \rightarrow D$. If B and C are the relevant attributes, [Sa2] would compute two extension joins, one from BCD alone and the other from AB and AC. However, taking the usual construction of maximal objects, we would get the one, cyclic, maximal object consisting of all three relations, by starting with the object AB. The reader may judge if the connection between B and C through A should be considered on a par with the connection in the single relation BCD. There seem to be arguments on both sides.

References

[A] Aho, A. V., private communication, June, 1981.

[ABU] Aho, A. V., C. Beeri, and J. D. Ullman, "The theory of joins in relational databases," *ACM Transactions on Database Systems* 4:3 (1979), pp. 297–314.

[ASU1] Aho, A. V., Y. Sagiv, and J. D. Ullman, "Equivalence of relational expressions," *SIAM J. Computing* 8:2 (1979), pp. 218–246.

[ASU2] Aho, A. V., Y. Sagiv, and J. D. Ullman, "Efficient optimization of a class of relational expressions," *ACM Transactions on Database Systems* 4:4 (1979), pp. 435–454.

[AP] Atenzi, P. and D. S. Parker, "Properties of acyclic database schemes: an analysis," *Proc. XP/2 Conf.*, June, 1981.

[B*] Beeri, C., R. Fagin, A. O. Mendelzon, D. Maier, J. D. Ullman, and M. Yannakakis, "Properties of acyclic database schemes," *Proc. Thirteenth Annual ACM Symposium on the Theory of Computing*, pp. 355–362, 1981.

[B] Bernstein, P. A., "Synthesizing third normal form relations from functional dependencies," *ACM Transactions on Database Systems* 1:4 (1976), pp. 277–298.

[BG] Bernstein, P. A. and N. Goodman, "What does Boyce-Codd normal form do?," *Proc. International Conference on Very Large Data Bases*, pp. 245–259, 1980.

[Cha] Chang, C.-L., "Finding missing joins for incomplete queries in relational databases," RJ 2145, IBM, San Jose, 1980.

[Che] Chen, P. P., "The entity-relationship model: toward a unified view of data," *ACM Transactions on Database Systems* 1:1 (1976), pp. 9–36.

[Co] Codd, E. F., "A relational model for large shared data banks," *Comm. ACM* 13:6 (1970), pp. 377–387.

[C*] Codd, E. F., R. S. Arnold, J.-M. Cadiou, C.-L. Chang, and N. Roussopolous, "Rendezvous version I: and experimental English language query formulation system for casual users of relational databases," RJ2144, IBM, San Jose, 1978.

[DSK] Dell'Orco, P., V. N. Spadavecchio, and M. King, "Using knowledge of a database world in interpreting natural language queries," *Proc. 1977 IFIP Congress*, North Holland, Amsterdam.

[F] Fagin, R., "On notions of acyclicity in databases and join dependencies," unpublished memorandum, IBM, San Jose, Calif.

[FMU] Fagin, R., A. O. Mendelzon, and J. D. Ullman, "A simplified universal relation assumption and its properties," RJ2900, IBM, San Jose, 1980.

[GW] Greenblatt, D. and J. Waxman, "A study of three database query languages," in *Database: Improving Usability and Responsiveness* (B. Schneiderman, ed.), Academic Press, New York, 1978.

[HLY] Honeyman, P., R. E. Ladner, and M. Yannakakis, "Testing the universal instance assumption," *Inf. Proc. Letters*, 10:1 (1980), pp. 14–19.

[Ke] Kent, W., "Consequences of assuming a universal relation," IBM Technical report, Dec., 1979, to appear in *TODS*.

[Kl] Klug, A., "Inequality tableaux," to appear in *JACM*.

[Ko] Korth, H. F., "System/U: a progress report," *Proc. XP/2 Conf.*, June, 1981.

[KU] Korth, H. F. and J. D. Ullman, "SYSTEM/U: a database system based on the universal relation assumption," *Proc. XP1 Conference*, Stonybrook, N. Y., June, 1980.

[L] Lien, Y. E., "On the equivalence of database models," to appear in *JACM*.

[Ma] Maier, D., "Discarding the universal instance assumption: preliminary results," *Proc. XP1 Conference*, Stonybrook, N. Y., June, 1980.

[MU1] Maier, D. and J. D. Ullman, "Maximal objects and the semantics of universal relation databases," TR-80-016, Dept. of C. S., SUNY, Stony Brook, N. Y., 1980.

[MU2] Maier, D. and J. D. Ullman, "Connections in acyclic hypergraphs," STAN-CS-853, Dept. of C. S., Stanford Univ., Stanford, Calif., 1981.

[Mc] McCarthy, W. E., "An entity-relationship view of accounting models," *The Accounting Review* 54:4 (1979), pp. 667-686.

[Mo] Moore, R. C., "Handling complex queries in a distributed database," Tech. Note 170, Artificial Intelligence, SRI Intl., Menlo Park, Calif., 1979.

[O] Osborn, S. L., "Towards a universal relation interface," *Proc. International Conference on Very Large Data Bases*, pp. 52-60, 1979.

[Sa1] Sagiv, Y., "Can we use the universal instance assumption without using nulls?," *ACM SIGMOD International Symposium on Management of Data*, pp. 108-120, 1981.

[Sa2] Sagiv, Y., "A characterization of globally consistent databases and their correct access paths," unpublished memorandum, Univ. of Illinois, Dept. of C. S., 1981.

[SY] Sagiv, Y. and M. Yannakakis, "Equivalences among relational expressions with the union and difference operators," *J. ACM* 27:4 (1980), pp. 633-655.

[Sc] Sciore, E., "Null values, updates, and normalization in relational databases," doctoral dissertation, Princeton Univ., Princeton, N. J., 1980.

[S*] Stonebraker, M., E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Transactions on Database Systems* 1:3 (1976), pp. 189-222.

[U] Ullman, J. D., *Principles of Database Systems*, Computer Science Press, Potomac, Md., 1980.

[WY] Wong, E. and K. Youssefi, "Decomposition—a strategy for query processing," *ACM Transactions on Database Systems* 1:3 (1976), pp. 223-241.

[Y] Yannakakis, M., "Algorithms for acyclic database schemes," unpublished memorandum, Bell Laboratories, Murray Hill, N. J., 1981.

PI *Confer-*

atabases,"

t. of C. S.,

ng Review

, Artificial

:e on Very

I SIGMOD

:ss paths,"

union and

ssertation,

INGRES,"

1980.

ransactions

, Bell Lab-